## Title
IRIS: Illustrative Rendering for Integral Surfaces



## Journal
IEEE Transactions on Visualization and Computer Graphics, 16

## Authors
Hummel, Mathias
Garth, Christoph
Hamann, Bernd
et al.

## Publication Date
2010

Peer reviewed

# IRIS: Illustrative Rendering of Integral Surfaces

Mathias Hummel, Christoph Garth,
Bernd Hamann, *Member, IEEE*, Hans Hagen, *Member, IEEE*, and Kenneth I. Joy, *Member, IEEE*

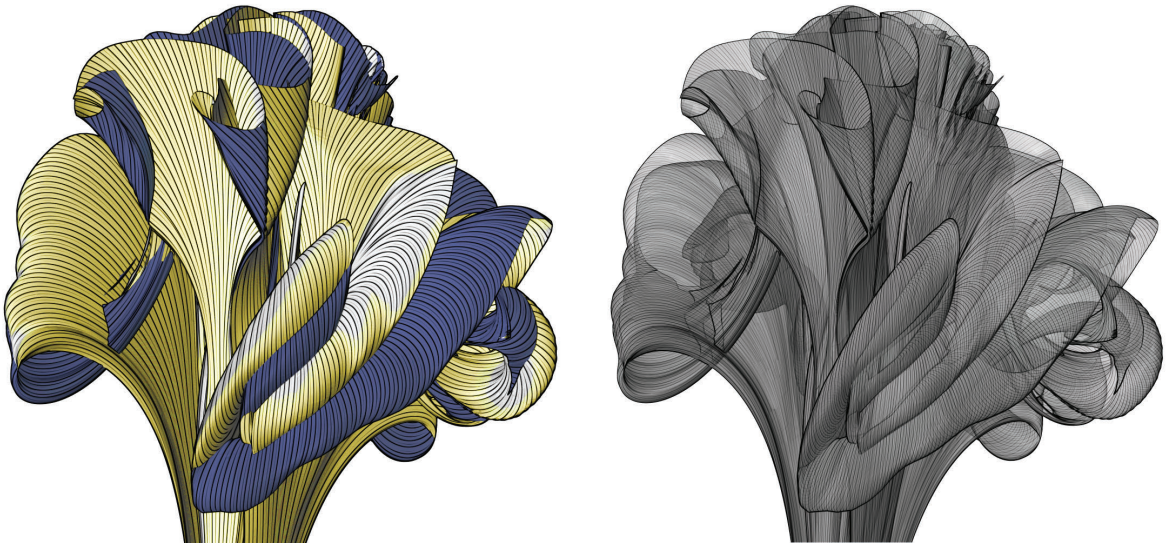


Fig. 1. A path surface generated from a turbulent jet dataset, rendered in two different styles using the framework proposed in this paper. In the left image, the surface is opaque, and the front and back side are rendered with yellow and blue, respectively. An adaptive stripe pattern visualizes individual pathlines on the surface and provides the orientation of the flow. On the right, the surface is rendered transparently with a denser stripes to give a hatching-like appearance. Both figures emphasize surface silhouettes for better distinction of individual surface layers.

**Abstract**—Integral surfaces are ideal tools to illustrate vector fields and fluid flow structures. However, these surfaces can be visually complex and exhibit difficult geometric properties, owing to strong stretching, shearing and folding of the flow from which they are derived. Many techniques for non-photorealistic rendering have been presented previously. It is, however, unclear how these techniques can be applied to integral surfaces. In this paper, we examine how transparency and texturing techniques can be used with integral surfaces to convey both shape and directional information. We present a rendering pipeline that combines these techniques aimed at faithfully and accurately representing integral surfaces while improving visualization insight. The presented pipeline is implemented directly on the GPU, providing real-time interaction for all rendering modes, and does not require expensive preprocessing of integral surfaces after computation.

**Index Terms**—Flow visualization, integral surfaces, illustrative rendering.

## 1 Introduction

Integral curves have a long standing tradition in vector field visualization as a powerful tool for providing insight into complex vector fields. They are built on the intuition of moving particles and the representation of their trajectories. A number of different variants exist; while streamlines and pathlines directly depict single particle trajectories, other curves visualize the evolution of particles that are seeded coherently in space (time lines) or time (streak lines). These curves imitate experimental flow visualization and correspond to smoke or dye released into a flow field. Generalizing on these concepts, integral surfaces extend the depiction by one additional dimension. Stream and path surfaces aim to show the evolution of a line of particles, seeded simultaneously, over its entire lifetime. These surfaces have been shown to provide great illustrative capabilities and much improved visualization over simple integral curves, and increase the visual insight into flow structures encountered during their evolution. Time surfaces increase the dimensionality further by showing the evolution of a two-dimensional sheet of particles. Finally, streak surfaces borrow from both path surfaces and time surfaces by portraying an evolving sheet of particles that grows during the evolution at a seeding curve as new particles are added to the surface. They are analogous to streak lines in that they originate from wind tunnel experiments with line-shaped nozzles and are therefore, in a sense, a very natural surface visualization primitive for time-varying flows.

In recent years, several new algorithms have been proposed for the computation of such integral surfaces, and techniques are now available that address a wide spectrum of visualization scenarios from real-time interaction and computation for smaller datasets using GPUs to very-complex large and time-dependent datasets using parallel algorithms. While surface computation is already quite complex, using

- *Mathias Hummel and Hans Hagen are with the University of Kaiserslautern, E-mail: {m_hummel|hagen}@informatik.uni-kl.de.*
- *Christoph Garth, Bernd Hamann and Kenneth I. Joy are with the Institute of Data Analysis and Visualization at the University of California, Davis, E-mail: {cgarth|hamann|kijoy}@ucdavis.edu.*

*Manuscript received 31 March 2010; accepted 1 August 2010; posted online 24 October 2010; mailed on 16 October 2010.*
*For information on obtaining reprints of this article, please send email to: tvcg@computer.org.*

integral surfaces for effective visualization can be quite difficult. Such surfaces often have a very high visual complexity (see e.g. Figure 1) that results from the shearing, twisting, and curling of the flow behavior they capture and describe. Thus, care must be taken when combining transparent rendering, texture mapping, and other illustrative techniques to preserve or enhance the understanding of the flow as conveyed through the surface. Different rendering and illustration approaches have been proposed previously, but as of yet it remains unclear which of these choices systematically work well for general integral surfaces, and how different techniques can be effectively and efficiently combined.

In this paper, we address the issues of transparency and texture mapping on integral surfaces by examining and adapting several existing visualization techniques (Sections 3 and 4). Furthermore, we present a rendering framework that combines these with other approaches from the field of illustrative rendering, described in Section 5. The system we describe is fully interactive, and all visualizations can be generated without laborious preprocessing. Our framework can thus be coupled with both interactive and non-interactive computation techniques in static or dynamic settings. We demonstrate the resulting visualization in application to complex examples from CFD simulation in Section 6 and briefly evaluate our results (Section 7), before we conclude in Section 8.

The benefits of the methods we discuss here with respect to integral surface visualization are twofold. First, the methods we describe are able to convey the full information contained in an integral surface by providing solutions to the problems of occlusion, complex three-dimensional structure, flow orientation, and dynamics. Second, by providing a framework that combines the different approaches, the complexity of choosing a specific visualization style is vastly reduced, and makes integral surface visualization accessible to visualization users.

## 2 Concepts and Related Work

Before we survey previous work on integral surface visualization, we briefly describe the basic concepts underlying integral surfaces as applied to flow visualization.

### 2.1 Basic Setting

If $v(t,x)$ is a (possibly time-dependent) three-dimensional vector field that describes a flow, then an *integral curve* $I$ of $v$ is the solution to the ordinary differential equation

$$I'(t) = v(t,I(t)), \qquad \text{and } I(t_0) = x_0, \tag{1}$$

where $v(t,x)$ is the vector at time $t$ and location $x$.

Technically, it is a curve that originates at a point $(t_0,x_0)$ and is tangent to the vector field at every point over time. Intuitively, it describes the path of a massless particle that is advected by $v$. In the typical case that $v$ is given in discrete form (e.g. as an interpolated variable on regular or unstructured grids), such integral curves can be approximated using numerical integration techniques. In the case where $v$ is independent of time, such integral curves are called *streamlines*, and *pathlines* in the time-dependent case.

An integral surface is the union of the trajectories of a one or two-dimensional family of integral curves, originating from a common seed curve or surface. Three specific instances of such surfaces are commonly distinguished:

- A *path surface* $P$ originates from a one-dimensional seed curve. The surface consists of the positions of all particles during their entire lifetime.
- A *time surface* $T$ is a two-dimensional family of integral curves that originate from a common seed surface, or alternatively, the surface formed by a dense set of particles that are located on the seed surface at the initial time and jointly traverse the flow.
- A *streak surface* $S$ is the union of all particles emanating continuously over time from a common seed curve and move with the flow from the time of seeding.

If $v$ is not time-dependent, a path surface is customarily labelled *stream surface* in analogy to integral curves. Furthermore, streak surfaces and stream surfaces are identical in this case. In this paper, we will generally use the term path surface, however, all discussion applies equally to stream surfaces.

Integral surfaces possess a natural parameterization. For path surfaces, it is given by the parameter $s$ that indicates the starting location on the seed curve, and the advection time $t$ of the corresponding particle to reach the given surface point. Lines of constant $s$-parameter are hence pathlines, and constant $t$-lines are called *time lines*. For streak surfaces, the situation is similar, but $s$-lines are *streaklines*. Time surfaces directly inherit the parameterization of their seed surface, i.e. the parameters (typically called $u$ and $v$) at each particle on the time surface correspond to the parameter of its seed location.

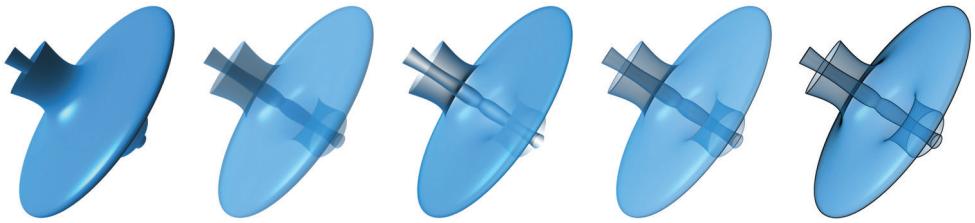
After establishing these basic notions, we will next briefly consider previous and related work on integral surfaces.

### 2.2 Integral Surface Generation

Integral surfaces were first investigated by Hultquist [17], who proposed a stream surface algorithm that propagates a *front* of particles forming a surface through a flow. Particle trajectories are integrated as needed and triangulation of the surface is performed on-the-fly using a greedy approach. Divergence and convergence of particles in the advancing front is treated using a simple distance criterion that inserts and removes particles to ensure a balanced resolution of the surface. Advanced visualization of integral surfaces was introduced by Löffelmann [26] in proposing texture mapping of arrows on stream surfaces (*stream arrows*), with the goal of conveying the local direction of the flow. This original work did not address stretching and divergence of the surface, which distorts the parameterization and consequently can result in very large or small arrows. The same authors subsequently addressed this by a regular, hierarchical tiling of texture space that results in adjusted arrows [25]. However, stream arrows are rarely used in integral surface visualization from CFD data due to the high visual complexity of the resulting surfaces.

Garth et al. [11] built on the work of Hultquist by employing arc-length particle propagation and additional curvature-based front refinement, which results in a better surface triangulation if the surface strongly shears, twists, or folds. They also considered visualization options such as color mapping of vector field-related variables going beyond straightforward surface rendering. A different computational strategy was employed by van Wijk [33], who reformulated stream surfaces as isosurfaces; however, his method requires increased computational effort to advect a boundary-defined scalar field throughout the flow domain. Scheuermann et al. [30] presented a method for tetrahedral meshes that solves the surface integration exactly per tetrahedron. Improving visualization [23], Laramee et al. employed the *Image-Space Advection* technique [24] to generate a visual impression of the flow direction on the surface. This depiction is naturally resolution-independent, but does require costly computation and a full representation of the vector field on the surface.

More recently, Garth et al. [10] replaced the advancing front paradigm by an incremental time line approximation scheme, allowing them to keep particle integration localized in time. They applied this algorithm to compute stream surfaces and path surfaces in large and time-varying CFD datasets. Using a GPU-based approach, Schafhitzel et al. [29] presented a point-based algorithm that does not compute an explicit mesh representation but rather uses a very dense set of particles, advected at interactive speeds, in combination with point-based rendering. Recently, Krishnan et al. [21], Bürger et al. [5] and von Funck et al. [34] presented approaches for time and streak surface computation. While the former authors focused on the CPU treatment of large CFD datasets, the latter designed their approach for GPUs with the aim of real-time visualization for smaller datasets. All three papers present various visualization options, including striped textures and advanced transparency though depth peeling (see e.g. [1]), but do not discuss these visualization choices and their realization in detail. The intent of this work is in part to adopt a systematic approach to integral surface visualization by discussing available visualization choices



(a) no transparency (b) constant (c) angle-based (d) normal-variation (e) with silhouettes

Fig. 2. A comparison of different transparent rendering styles. Images (c) through (e) show the effect of view-dependent transparency modulation.

in detail, and to describe their implementation in sufficient detail to be easily reproducible.

### 2.3 Illustrative Rendering and Integral Surfaces

Computer graphics offers many techniques to realistically render objects in static and animated representation, and to create new scenes under the same environmental conditions; for an overview, we refer the reader to the works by Gooch and Gooch [13] and Strothotte and Schlechtweg [32]. For non-photorealistic rendering, approaches have been presented to reproduce numerous artistic techniques, such as tone shading [12], pencil drawing [4], hatching [27], or ink drawing [31]. In the context of integral surfaces, however, artistic representation plays a secondary role to an accurate depiction the structure of the flow as captured by the surface. For example, while hatching techniques providing shape cues for a depicted surface, the hatching pattern introduces directional information which is at risk of being confused with flow direction. Gorla et al. [14] study the effect of directional surface patterns on shape perception.

The use and combination of non-photorealistic techniques to highlight and illustrate specific aspects of a dataset has been examined in detail in its application to volume rendering, where similar constraints apply. Here, Ebert and Rheingans [8] present several illustrative techniques such as boundary enhancement and sketch lines which enhance structures and add depth and orientation cues. Csebfalvi et al. [6] visualize object contours based on the magnitude of local gradients as well as on the angle between viewing direction and gradient vector using depth-shaded maximum intensity projection. Krüger et al. [22] use an interactive magic lens based on traditional illustration techniques to selectively vary the transparency in the visualization of iso-surfaces; this technique is termed *ClearView*.

In this context, one goal of this work is to apply and adapt specific techniques from illustrative rendering to the specific case of integral surface visualization. Evaluating the quite significant body of work on illustrative techniques for this scenario is beyond the scope of this work; rather, we focus on two core aspects of integral surface rendering: transparency and texturing. This choice is based on the authors' observation of typical problems that complicate integral surface visualization, and is discussed in more detail in Sections 3 and 4 below.

Furthermore, we consider the following characteristics to select techniques. First, we observe that integral surfaces can imply a significant computational burden in the presence of large and complex flow data sets. The surface representations resulting from such data can be comprised of millions of triangles and take minutes to hours to compute (cf. [10, 21]), and interaction with the surface in a near real-time setting – possibly even during the surface computation – is highly desirable. For less complex data, the recent work of Bürger et al. [5] describes a real-time computation approach that leverages the computing power of GPUs, and we aim at retaining the applicability of the methods described in this paper in such a scenario. Similarly, the dynamic and evolving nature of time and streak surfaces attractively captures the temporal characteristics of flows; as such, the ability to animate integral surfaces is pertinent to our considerations.

In the following sections, we describe approaches to transparency and texturing that fulfill these requirements.

## 3 Transparency

Due to folding, twisting, and shearing of the flow traversed by them, integral surfaces often possess a very high depth complexity and one often finds that large parts of the surface are occluded by the surface's outer layer in an opaque depiction of the surface. Introducing transparency into the rendering can alleviate this; however, the number of layers is often so large that the straightforward choice of constant transparency produces unsatisfactory results. A low constant transparency over the entire surface typically results in good visibility of the outer layers, while the inner layers are occluded. Conversely, if the constant transparency is high to reveal the inner layers, the outer layers are hard to identify. As discussed previously by Diepstraten et al. [7] among others, transparency in illustrations often applies the *100-percent-rule*, stating that transparency should fall off towards the edges of an object. This results in a non-uniform decrease of the transparency of layers as the depicted surface curves away from the viewer.

The same authors propose an object-space algorithm to achieve this by varying the transparency of a surface point as a function of its distance to its outline. The outlines of an object projected onto a 2D screen consist of silhouettes lines (see also Section 3.4), and thus the distance computation entails the extraction of an explicit description of the view-dependent silhouette lines. To this purpose, an object space approach is proposed that is too costly for large surfaces with several millions of triangles. Moreover, this technique does not provide insight into the curvature of the transparent surface. Taking a different approach, the methods proposed by Kindlmann et al. [20] and Hadwiger et al. [15] for direct volume rendering and iso-surface rendering vary surface properties in dependence of the principal surface curvatures and are used to emphasize surface detail such as ridges and valleys. Thus, using such curvature measures to influence transparency of an integral surface seems appealing. Judd et al. [19] used *view-dependent curvature* to extract so-called *apparent ridges*. However, none of these methods address transparency directly, and direct application to our problem would require the computation of high-quality curvature measures. For the interactive visualization of large triangle meshes such as integral surfaces, we consider such approaches too computationally expensive.

We instead propose two simple measures for transparency variation that are cheap to compute and give very good results.

### 3.1 Angle-Based Transparency

If $n$ is the surface normal at the considered point and $v$ is the normalized view vector, then choosing the *view-dependent transparency*

$$\alpha_{\text{view}} := \frac{2}{\pi} \arccos(n \cdot v)$$

varies the transparency with the angle between $n$ and $v$. This has the effect that surface regions orthogonal to the viewer become more transparent, while regions where the surface curves towards or away from the viewer are more opaque. This decreases the transparency as the object silhouette is approached, and surface curvature is indicated indirectly by the image-space extent of the opacity gradient, as shown in Figures 3(a) and Figure 2(c). A drawback of this approach is the dependence of the transparency gradient on the curvature radius of the

surface. If the integral surface contains large, almost flat parts, their resulting high opacity can obscure the layers below.

### 3.2 Normal-Variation Transparency

The second approach we propose is related to the work of Kindlmann et al. [20] in controlling the transparency as a function of the surface curvature perpendicular to the viewer, as determined in image space. If $n(i,j)$ denotes the surface normal at a pixel $(i,j)$, we observe that the partial derivatives of the normal's $z$-component

$$\frac{\partial n_z}{\partial i} \quad \text{and} \quad \frac{\partial n_z}{\partial j}$$

provide a rough approximation of the curvature of the surface perpendicular to the image plane in view space. By letting

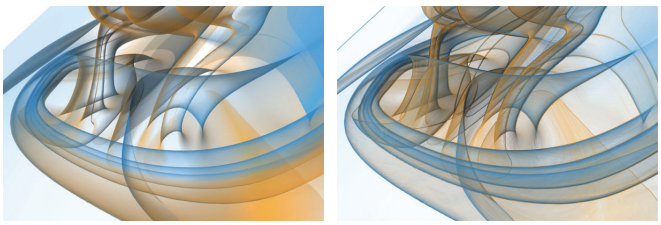
$$\alpha_{\text{view}} := \left( (\frac{\partial n_z}{\partial i})^2 + (\frac{\partial n_z}{\partial j})^2 \right)^{\frac{\gamma}{2}}, \tag{2}$$

assuming $\gamma = 1$ for now, we obtain a transparency function that is approximately proportional to the local surface curvature perpendicular to the viewer. As a result, the surface is more opaque where it curves away from the viewer and most transparent when it is oriented perpendicular to the viewer. Furthermore, this achieves the effect that for surface features with strong curvature such as creases or small features, the transparency is reduced, resulting in a better visual impression of such small details. Here, $\alpha_{\text{view}}$ is not exclusively dependent on the view direction, such that strongly curved surface features can be well identified even if viewed frontally (see Figure 2(d)). We note that $\alpha_{\text{view}}$ is not necessarily limited to the unit interval, and must be clamped before $\gamma$ is applied.

Secondly, for surface regions curving away from the viewer, $n_z$ varies fastest as the silhouette is approached, leading to quickly increasing transparency towards the boundary, as opposed to a slow gradation using the angle-based transparency. In the context of integral surface visualization, this aspect is important since it allows a clear visual understanding of nested flow tubes that are generated by the flow rolling the surface up into a nested set of tubes. Since curvature increases for the inner tubes, they are more prominently visible in the resulting image. This phenomenon and the visual differences of normal-variation transparency over angle-based transparency are illustrated in Figure 3.

The parameter $\gamma$ in Equation 2, selected over the unit interval, allows a smooth control of the strength of the normal variation transparency, where we provide selection of $\gamma$ over $[0,1]$. Larger values emphasize the surface silhouettes and provide little additional insight. It is our experience that controlling $\alpha_{\text{view}}$ exponentially provides more intuitive control over the effect strength than e.g. linear scaling. Furthermore, we found it helpful to constrain the overall resulting transparency range to a user-adjustable interval $[\alpha_{\min}, \alpha_{\max}]$ through

$$\alpha = (1 - \alpha_{\text{view}}) \cdot \alpha_{\min} + \alpha_{\text{view}} \cdot \alpha_{\max}.$$



(a) Angle-based (b) Normal-variation

Fig. 3. A comparison of angle-based and normal-variation view-dependent transparency rendering. In (b), the radius of the tubes affects the transparency; the thinnest tube is most opaque. Furthermore, the opacity gradient in (b) is stronger than in (a).

### 3.3 Window transparency

Despite obtaining very good results for most integral surface visualization scenarios, we have nevertheless observed the occasional need to selectively increase the transparency of the visualized surface in certain regions. In the setting of iso-surfaces, Krüger et al. [22], inspired by the earlier work of Bier et al. [2], presented an innovative approach to allow a user to selectively vary the transparency of an occluding surface through a user-controlled window. We adopt a similar approach: we modulate the overall transparency of the rendering as a function of the surface pixel position in image space. Typically, we decrease this *window transparency* $\alpha_{\text{window}}$ smoothly with the distance to a specified point in image space. This allows the easy creation of an interaction with "windows" that allow seeing through the context provided by an outer surface layer to reveal otherwise occluded details (see Figure 11).

### 3.4 Silhouettes

Silhouettes, described e.g. by Gooch et al. [12], are a popular technique for non-photorealistic rendering. Through visually emphasizing the transition between front- and back-facing surface layers, object shape is revealed, and sharp surface features such as creases are highlighted. In transparent depictions, distinguishing layers can be difficult. Here, silhouettes reveal the layer boundaries and provide hints at the surface's shape. Corresponding rendering approaches can be mainly divided into the object-space, image-space, and hybrid categories (cf. [16, 18]). Isenberg et al. [18] recommend using an image space technique for achieving interactive frame rates with huge or animated data sets. Object-space and hybrid algorithms rely on processing the mesh and thus we exclude them from consideration due to the high effort required for the large integral surfaces meshes we consider here.

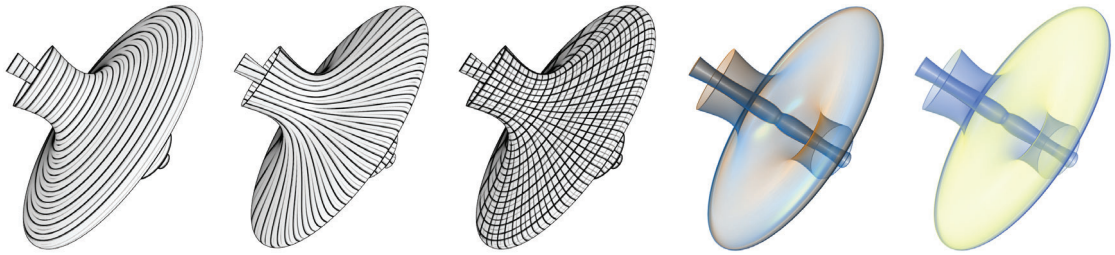
In the framework presented here, we make use of an image space silhouette detection algorithm that is described in Section 5.

## 4 Textures

When used correctly, textures are powerful tools that, in application to integral surfaces, serve a dual purpose. First, they can be employed to enhance shape perception, providing a better visual comprehension of the complex surface shape. Second, and specific to integral surfaces, they can indicate flow orientation on the surface, or specific streamlines, pathlines, streak lines, or time lines on a surface. Integral surfaces provide a natural surface parametrization by virtue of their construction (see Section 2). For path surfaces, unique $s$ and $t$ parameters correspond directly to pathlines and timelines, respectively. Time surfaces can inherit their parameterization from a parametric seed surface, and streak surfaces present a hybrid, where again constant $t$ indicates time lines and fixing $s$ provides streak lines. By carrying this parameterization from the surface computation stage to the rendering stage, textures can thus be applied to highlight flow behavior on the surface.

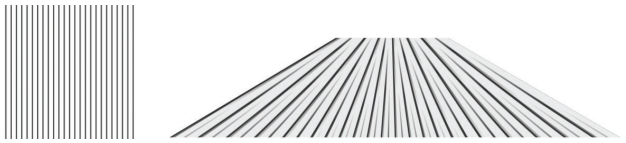
Regarding shape enhancement, several illustrative rendering techniques approximate a hatching-type depiction to improve shape perception (see e.g. the work of Freudenberg et al. [9] and Strothotte and Schlechtweg [32]). However, in this context, this introduction of directional information is at risk of being confused with flow direction. Thus, when applying such techniques, the pattern must be oriented along the existing parameterization.

Unfortunately, the natural integral surface parameterization is subject to strong and possibly anisotropic distortion that reflects the convergence or divergence of neighboring particles traversing the flow. In typical integral surfaces, it is not uncommon that the surface is stretched by a factor of 1000 or greater. Thus, if the intent is to highlight individual flow lines through a straightforward application of a stripe texture, large gaps may appear between stripes, and stripes grow strongly in width, thus obscuring the flow depiction in such areas (see e.g. Figure 6(a)).



(a) $s$-direction (b) $t$-direction (c) $s$- and $t$-direction (d) two-sided shading (e) cool-warm shading

Fig. 4. The adaptive pattern has a constant resolution in image space, regardless of the variation of the texture coordinates across the surface, as illustrated in (a), (b), and (c). Figure (d) demonstrates the effect of shading front and back sides of the surface in different colors, while (e) depicts cool-warm shading that substitutes color variation for surface brightness change for illumination.



(a) Stripe pattern (b) Perspective view, adaptive stripe pattern

Fig. 5. Perspective distortion of texture density is addressed by adaptive pattern evaluation.

### 4.1 Adaptive Patterns

Freudenberg et al. [9] proposed an approach for real-time surface hatching by using a mipmap-like technique, so called *hatch maps*. In conventional mipmapping, the texture to be mapped onto an object is replaced by a stack of textures, where lower levels represents down-sampled versions of the highest-resolution texture. Mipmapping then selects an appropriate level to sample from the stack by taking into account the image space variation of the texture coordinates, such that a roughly one-to-one reproduction of texels to pixels is achieved and smoothly blends between consecutive levels to avoid image discontinuities. Freudenberg et al. repurpose this mechanism by loading the stack with successively smaller images that contain lines exactly one pixel wide. Thus, they achieve a approximately constant image space line density, giving the impression of hatching.

Our texturing approach is based on a similar idea; however, instead of using a finite set of textures with different resolutions, we reuse a single texture or pattern and adjust the sampling frequency to yield approximately constant image space pattern density. Furthermore, we compensate for highly anisotropic stretching by determining the sampling frequency independently for the parameter directions $s$ and $t$.

The variation $\lambda_{s,t}$ in texture coordinate in image space at a pixel $(i, j)$ is determined by the image-space partial derivative of the texture coordinates $s$ and $t$ at $(i, j)$ as

$$\lambda_s(i,j) = \sqrt{\left(\frac{\partial s}{\partial i}\right)^2 + \left(\frac{\partial s}{\partial j}\right)^2} \tag{3}$$

and

$$\lambda_t(i,j) = \sqrt{\left(\frac{\partial t}{\partial i}\right)^2 + \left(\frac{\partial t}{\partial j}\right)^2} \tag{4}$$

If either of $\lambda_{s,t}$ doubles, the pattern frequency in the corresponding direction must be halved to yield the same image space frequency. If the pattern is described by a function $P(s,t)$ over the unit square, we determine two integer resolution levels $l_s$ and $l_t$ via

$$l_s = \log_2 \lambda_s \qquad \text{and} \qquad l_t = \log_2 \lambda_t,$$

and define the frequency-adjusted pattern $\hat{P}$ by evaluation of $P$ with correspondingly compensated frequency through

$$\hat{P}_{l_s,l_t}(s,t) \;:=\; P(s \cdot 2^{-l_s}, t \cdot 2^{-l_t}).$$

Since resolution levels are discretely defined, we apply bilinear interpolation between neighboring resolution levels to obtain a smooth pattern frequency in image space:

$$\begin{aligned} c(s,t) = (1-\tilde{l}_s) \cdot \Big( (1-\tilde{l}_t) \cdot \hat{P}_{\lfloor l_s \rfloor, \lfloor l_t \rfloor}(s,t) + \tilde{l}_t \cdot \hat{P}_{\lfloor l_s \rfloor, \lceil l_t \rceil}(s,t) \Big) \\ + \tilde{l}_s \cdot \Big( (1-\tilde{l}_t) \cdot \hat{P}_{\lceil l_s \rceil, \lfloor l_t \rfloor}(s,t) + \tilde{l}_t \cdot \hat{P}_{\lceil l_s \rceil, \lceil l_t \rceil}(s,t) \Big) \end{aligned} \tag{5}$$

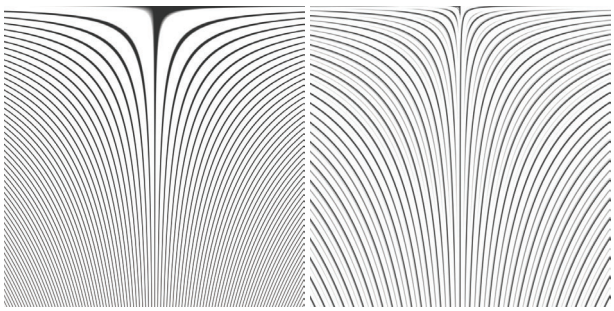
where

$$\tilde{l}_s = l_s - \lfloor l_s \rfloor, \quad \tilde{l}_t = l_t - \lfloor l_t \rfloor$$

denote the fractional parts of $l_s$ and $l_t$, respectively.

This mapping counters the effect of surface stretching on the pattern representation, and additionally perspective foreshortening (similar to [9]), by adapting the scale of the pattern reproduction (see Figures 5 and 6). As demonstrated below in Section 6, this allows the effective use of stripe textures and other texture variations to highlight surface shape and distortion as well as flow behavior directly. While Equations 3 and 4 seem difficult to evaluate at first glance, such evaluation can leverage built-in functionality of the rendering system as discussed below in Section 5 and is actually cheap to evaluate. Note that for the case described above, the pattern can either be procedural (such as stripes, see Figure 4) or originate from an arbitrary texture image.

Patterns and textures can be applied in various ways to modulate both opacity and color of the integral surface to achieve specific visualization styles. We will discuss a number of examples in Section 6. The rendering pipeline we describe in Section 5 specifically focuses on the straightforward cases of modulating surface transparency additively and multiplicatively, which covers a large variety of use cases.



(a) Regular stripe pattern. (b) Adaptive stripe pattern.

Fig. 6. Strong anisotropic surface texture coordinate stretching is addressed by an adaptive pattern.

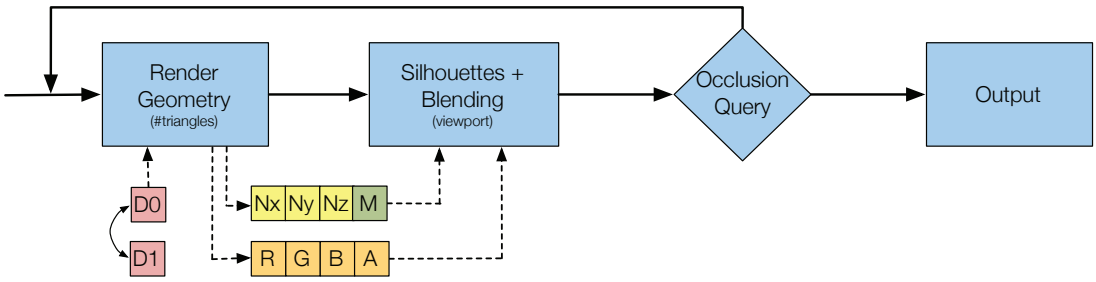


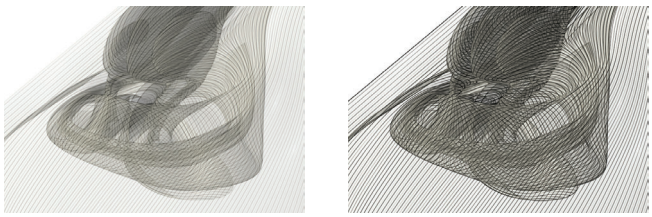

Fig. 8. Rendering pipeline overview.

## 5 Rendering Pipeline

In this section, we provide a description of our implementation of the integral surface rendering techniques discussed above. Our implementation is based on OpenGL, however, the concepts we make use of could be easily ported to DirectX. We make use of programmable shading via GLSL shader programs throughout the pipeline. As input for our techniques, we require for each frame a discrete representation of the integral surface (e.g. triangles or quadrilaterals), with a set of two texture coordinates associated with each vertex. This representation need not be connected and we explicitly accommodate some computation approaches (such as the method by Bürger et al. [5]) that generate surfaces as (partial) primitive soup. The normal-variation transparency approach described in Section 3.2 requires continuously varying normals over the mesh; in this case, the normal must be specified per vertex. If uniform or angle-based transparency are to be applied, face-based normals are sufficient. In this case, no preprocessing is required at all, and illustrative integral surfaces can be rendered during the computation phase. Additionally, our approach supports the addition of context geometry such as flow domain boundaries that are rendered and shaded independently from the integral surface.

Since our approach makes heavy use of transparency, a primary concern is the correct representation of transparent surfaces. Typically, there are two approaches to achieve correct transparency rendering at interactive speeds. The first, *depth sorting*, is based on sorting all elementary primitives by distance from the camera; primitives are then rendered from back to front with *over-blending* to ensure correct transparency. This approach is conceptually simple, but requires implementation on the GPU to achieve competitive performance. While this is not a significant problem, it suffers from complications with surfaces that are self-intersecting. Thus, we cannot apply it in this context since path surfaces often self-intersect.

Conversely, the *depth peeling* approach (see e.g. [1]) decomposes the rendering into layers of different depth. By rendering the primitives comprising the scene multiple times and discarding surface fragments that are closer to the viewer than those in the previous layer, one obtains incremental layers of depth. These layers are successively blended together to assemble the final image. This can be performed in both back-to-front order (using *over* blending) or in front-to-back order (using *under* blending). The peeling approach lends itself naturally to the image-based rendering techniques discussed above. Furthermore, since depth ordering is resolved per pixel, self-intersecting surfaces do not pose a problem. On the downside, this flexibility is balanced by the need to render a potentially large primitive set multiple times for a single frame.



(a) Multiplicative modulation: opacity is not increased by the pattern

(b) Additive modulation: the stripe pattern adds to the surface opacity

Fig. 7. Streamlines on a stream surface in the ellipsoid dataset are visualized by a stripe pattern.

### 5.1 Peeling Loop

For each frame, a depth peeling loop with on-the-fly front-to-back peeling is executed (for a detailed discussion, we refer the reader to the description by Bavoli and Myers [1]). Each iteration of the loop consists of two stages. The first stage (*peel stage*) computes a new depth layer of the surface based on the previous depth layer, and the second stage (*blend stage*) blends it into the framebuffer. We adaptively terminate the peeling loop if no pixels are rendered during the first stage, which is determined using an occlusion query for each loop iteration. The total number of iterations is thus one greater than the number of surface layers for the current viewpoint.

During the peel stage, we perform full surface shading, i.e. lighting and texture evaluation of the integral surface.

**Transparency** If required, the view-dependent transparency term $\alpha_{\text{view}}$ is directly computed from the normal vector of the rendered fragment; in the case of normal-variation transparency, the GLSL instructions `dFdx` and `dFdy` are used to evaluate the partial derivatives in Equation 2 directly and with little additional overhead. Otherwise, $\alpha_{\text{view}}$ is assigned a constant uniform value.

**Pattern or Texture** In the case of adaptive patterns, $l_s$ and $l_t$ are again computed using `dFdx` and `dFdy`, and Equation 5 can be directly evaluated, using either a procedural pattern that is directly evaluated in the shader or through corresponding texture lookups. Overall, we obtain texture color $c_{\text{tex}}$ and $\alpha_{\text{tex}}$.

**Lighting** The diffuse color of the surface is evaluated according to the specified lighting model; currently, we employ Phong and Gooch models. The result is the diffuse surface color $c_{\text{diffuse}}$. We also compute specular highlight terms $(c,\alpha)_{\text{spec}}$ if specified by the user; however, we keep diffuse and specular components separate to ensure correct blending of the highlights with the surface and texture colors.

**Combination** The final RGBA output $(c,\alpha)_{\text{peel}}$ of the peel pass is computed as

$$\begin{aligned} \alpha_{\text{final}} &= \alpha_{\text{view}} \cdot \alpha_{\text{tex}} + \alpha_{\text{specular}} \qquad \text{and} \\ c_{\text{final}} &= c_{\text{diffuse}} \cdot c_{\text{tex}} + c_{\text{specular}}, \end{aligned}$$

in the case where surface opacity should be multiplicatively modulated by the texture opacity. If additive modulation is desired, the alpha term changes to

$$\alpha_{\text{final}} = \alpha_{\text{view}} + \alpha_{\text{tex}} + \alpha_{\text{specular}}.$$

The final RGBA values are written to a color buffer and surface normals required in the blend stage are stored in a secondary floating-point RGBA buffer. Here, the secondary A-component contains a

mask to that indicates whether a pixel belongs to the integral surface or the context. This allows the blend stage operations to apply to surface pixels only and to not affect the context geometry pixels. Note that the depth information of the current surface layer is already stored into a separate depth texture that is required by the depth peeling algorithm.

In the blend stage, we determine the silhouette strength by applying a Sobel edge detection filter to both normal and depth buffers (as first described by [28]). Here, the mask is used to avoid generating silhouette pixels across surface-context and context-background pixels, by excluding pixels that have the mask flag set for one of the pixels contributing to the filter. Then, depending on the silhouette strength, the surface color is smoothly faded into a user-specified silhouette color.

We note silhouettes are essentially extracted twice – once per pair successive depth layers – by this approach, possibly resulting in inner silhouettes of increased width. However, since we use a relatively sharp and symmetric edge detection filter, this effect is reduced. While pixel-exact silhouettes are preferable, we have nevertheless opted to use this edge detection approach due to its purely image-space nature whose complexity is a function of the viewport size rather than the object-space mesh size, which can be very large in our case (cf. [18] for a more detailed discussion).

In our implementation, instead of blending directly to the output framebuffer, we make use of a floating-point color buffer for improved blending accuracy in the presence of many surface layers since slight banding artifacts can appear otherwise.

### 5.2 Performance

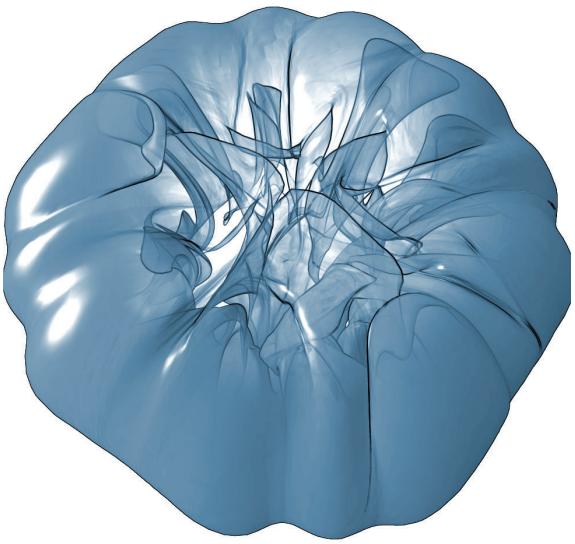
The rendering speed of the above pipeline is largely a function of the surface geometry size, and overall rendering time is dominated by the requirement to submit the surface for rendering multiple times during the depth peeling. For small to medium meshes of below ≈500K triangles with medium depth complexity of 10 layers or less, we achieve interactive frame rates exceeding 15fps on moderately current graphics hardware (NVidia GTX280). An exact quantification is difficult – and we do not attempt one in this paper – since the adaptive termination of the depth peeling implies that the number of peeling passes is a function of the depth complexity of the surface, the surface representation, and the currently chosen viewpoint. Larger meshes with many layers result in correspondingly smaller frame rates.

We experimented with a number of different implementation techniques, including fully deferred shading (cf. Saito and Takahashi [28]), but did not observe a significant variation in rendering speeds in our experiments. Again, we conclude that the geometry overhead from the multiple peeling passes dominates all other factors such as shader runtime or memory bandwidth.

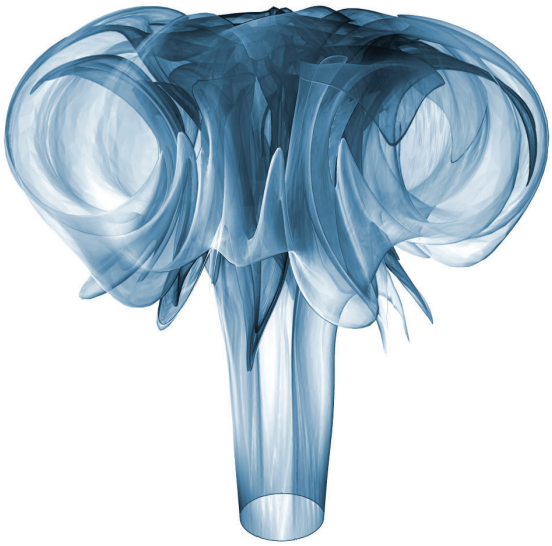
## 6 Examples

In the following, we briefly wish to discuss a number of sensible ways in which patterns and/or textures can be applied to enhance the visualization of integral surfaces.

While we did not consider lighting and color in the previous section, it can nevertheless play an important role in generating insightful integral surface visualizations. In general, we have found high-quality lighting, such as e.g. per-pixel evaluation of the commonly used *Phong* lighting model using multiple light sources, to be conducive to the impression of surface shape obtained by the viewer. However, approximately photorealistic lighting is not optimal in some situations. Especially in cases where the surface texture has strongly varying intensity (such as LIC-like images or stripes), the added variation through diffuse shading can lead to confusing results, since both shape and texture are encoded using only luminance information [35]. In these situations, it can be beneficial to adopt hue values such as found e.g. in *Gooch shading* [12], to convey either of the two channels of information. Furthermore, since integral surfaces are typically not closed, we have found it very helpful to choose different colors for the two sides of the surface. To provide example illustrations, we used the rendering pipeline described in Section 5 to produce renderings of several integral surfaces computed for different flow fields.



(a) Windowed transparency provides insight while preserving context.

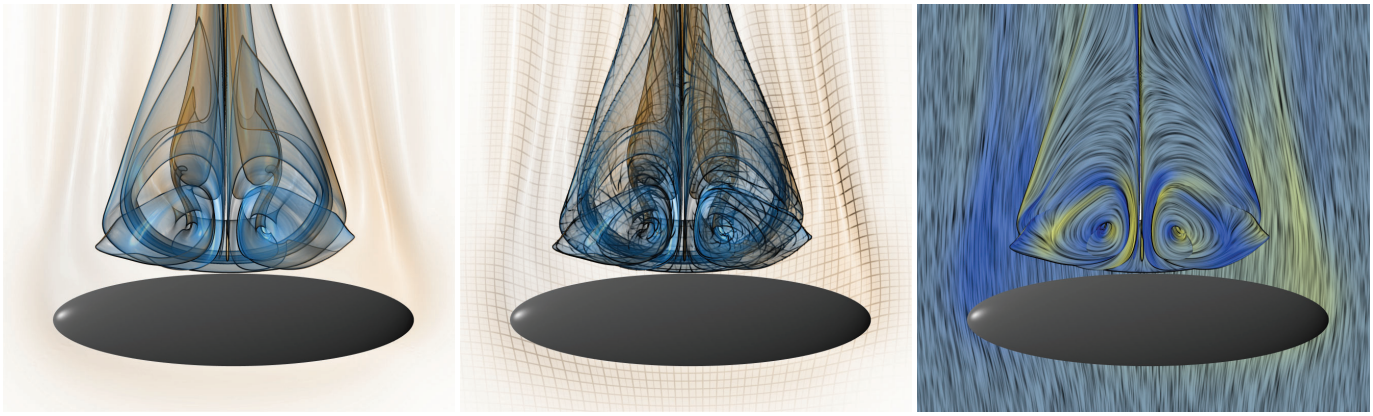


(b) Normal-variation transparency preserves folds.

Fig. 9. A rising plume streak surface is rendered using different styles.

Plume We applied normal variation based transparency to a streak surface in a dataset containing a rising plume (Fig. 9). The plume contains many strongly folded surface parts that result in sharp ridges. Here, normal-variation curvature is particularly effective in preserving the opacity of the folds. In figure 9(a), a transparency window is used to preserve context.

Ellipsoid Figures 10(a) and 10(b) show a stream surface of a flow field behind an ellipsoid (Figure 10). Normal-variation based transparency is used to reveal the shape of the entire surface including otherwise difficult to recognize inner tube-like structures. Recognition of these shapes is further facilitated by the application of silhouettes. The surface is rendered using different colors for front and back sides (blue and orange, respectively). Thus, the viewer can recognize areas where the surface reverses orientation. For Figure 10(b), an adaptive rectangular grid pattern was used to visualize both streamlines and timelines simultaneously. To avoid overwhelming the viewer with excessive lines, the texture modulates the surface opacity multiplicatively and is thus subject to transparency modulation. This causes the texture to be highlighted only on curved surface parts.



(a) Transparency based on normal variation conveys surface shape.

(b) A grid pattern modulating opacity multiplicatively shows both streamlines and timelines.

(c) Using a noise texture blurred in $t$-direction results in a LIC-like depiction of the flow.

Fig. 10. Flow behind an ellipsoid rendered using different styles.

Figures 7(a) and 7 demonstrate the effect of multiplicative versus additive opacity modulation. Constant opacity is used with a striped texture to visualize streamlines. With additive modulation, the streamlines are more clearly visible while multiplicative modulation causes slightly less occlusion.

Figure 10(c) shows a rendering featuring an effect similar to *Line Integral Convolution*, obtained by mapping a pre-convoluted (in $t$-direction) noise texture onto the surface using the adaptive pattern technique. The texture consists of noise blurred in $t$-direction. Cool-warm shading is used together with silhouettes to convey shape information.

Delta Wing Figure 11 shows renderings of a stream surface in the delta wing dataset. To visualize the flow direction on the surface, a stripe pattern along the $s$-direction is used with our adaptive pattern approach (Fig. 11 (a)). A user-defined window is used to restrict normal-variation based transparency to a small area. In Figure 11 (b), front and back side of the mesh are colored differently to indicate regions where the surface is flipped. Both figures apply windowed transparency and provide insight on the shape of otherwise hidden inner structures while preserving the context in the scene.

Furthermore, Figure 12 (a) illustrates strong normal variation transparency in combination with light silhouettes applied to the visualization of a stream surface traversing a vortex breakdown bubble in the same dataset. Even though the surface is quite complex, many layers and tube structures can be well identified. For the same surface, a visualization resembling a set of dense particle trajectories similar to those generated from dye advection-type methods (e.g. [36]) can be obtained with the adaptive pattern technique, shown in Figure 12) (b). Here, $s$-stripes that indicate individual streamlines on an otherwise opaque surface are further modulated in opacity and color to indicate direction and time. Note that even though the texture coordinates of the surface are highly distorted, stripes are somewhat evenly distributed.

## 7 Evaluation

While we have not attempted a systematic evaluation of the different illustrative styles discussed above, we have shown the resulting depictions to a number of collaborators from the flow simulation community. In the informal feedback we have gathered, the adaptive transparency was rated highly for providing improved insight into the inner surface structures while maintaining the context and shape of the surrounding layers. Here, the silhouettes were regarded important in determining layer boundaries. Furthermore, the additional shape cues provided through the adaptive patterns were determined useful to gain insight into aspects of the flow not conveyed by shape alone. In general, a more photorealistic look, including high-quality lighting, was generally preferred over more abstract depictions. While the feedback was largely positive, a more systematic study is indicated as future work.

## 8 Conclusion

We have discussed several rendering techniques with regard to their applicability for illustrative visualization of integral surfaces. The presented techniques are incorporated in an illustrative rendering framework for integral surface visualization. View-dependent transparency criteria provide improved visualization of deeply nested flow struc-

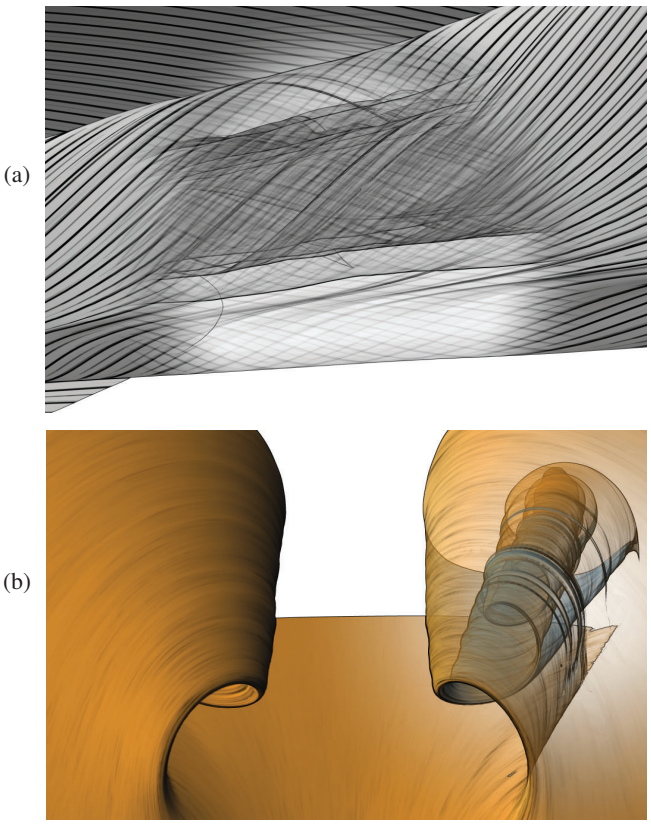



Fig. 11. Stream surface in the delta wing dataset, with windowed transparency. In (a), a stripe texture is used to visualize streamlines, and deeper layers of a vortex are visible. (b) shows a front view with two-sided surface coloring.

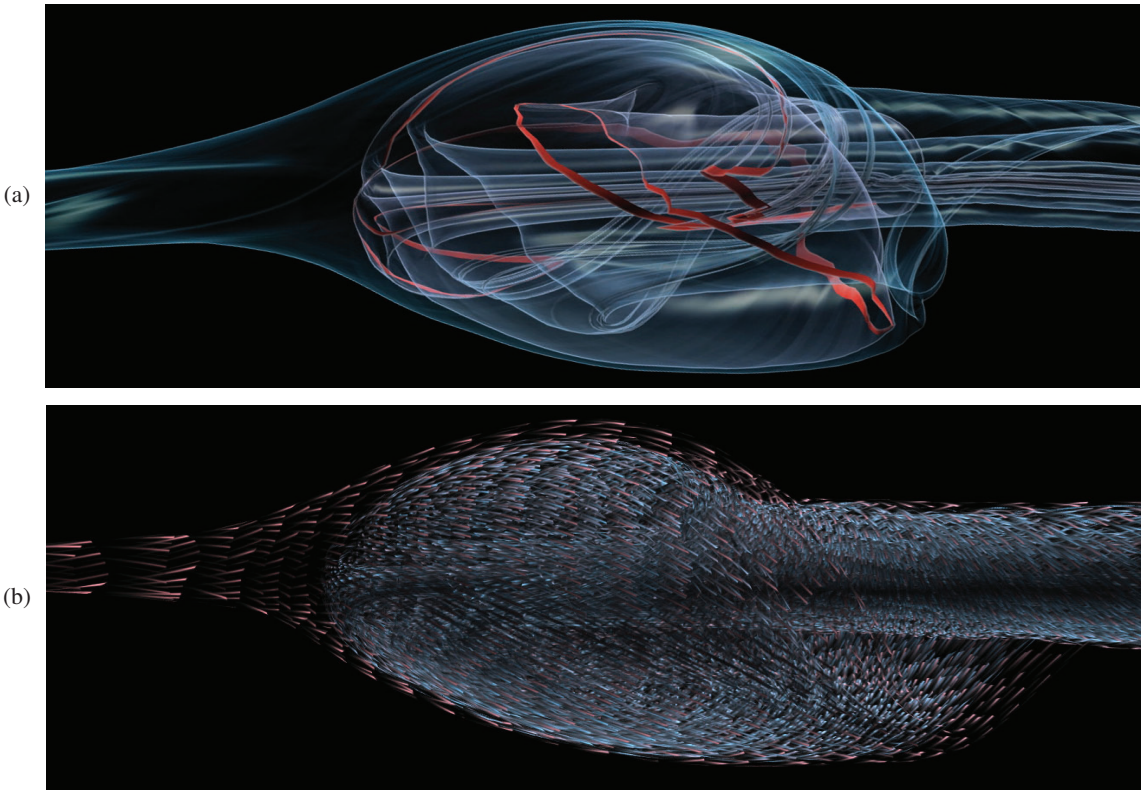



Fig. 12. A stream surface visualizes flow inside a vortex breakdown bubble. In (a), the surface is rendered with strong normal variation transparency and light silhouettes. The opaque red stripe illustrates the front of the surface. In (b), a modulated stripe texture conveys the impression of dense particles traces; here, flow direction is indicated by intensity modulation, and velocity is expressed as the length of the traces.

tures, and an adaptive pattern approach easily allows application of both shape-accentuating and flow-illustrating patterns and textures. Our framework is applicable to dynamic or animated surfaces. It does not require expensive preprocessing of the integral surface mesh, and can thus be applied to both interactive and exploratory settings for static as well as dynamic datasets. Furthermore, we have provided an in-depth overview of the combined realization of the presented rendering techniques in the form of a rendering framework, and have discussed specific steps in detail. We have demonstrated the capabilities of our framework on several examples involving very complex integral surfaces from CFD applications.

In the future, we wish to examine incorporating the concept of *style textures* (described by Bruckner and Gröller [3]) into our rendering pipeline to allow users to specify integral surface appearance by selecting a style. Furthermore, we wish to examine the efficient and effective mapping of glyphs onto the surface to allow users to annotate the surface. Last but not least, we plan to evaluate our approach through a formal user study.

## Acknowledgments

The authors wish to thank Markus Rütten from DLR Germany for the datasets used in this paper. We also thank our colleagues at the University of Kaiserslautern and at the Institute for Data Analysis and Visualization at UC Davis for discussion and support. This work was supported in part by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-FC02-06ER25780 through the Scientific Discovery through Advanced Computing (SciDAC) programs Visualization and Analytics Center for Enabling Technologies (VACET).

## References

[1] L. Bavoli and K. Myers. Order-independent transparency with dual depth peeling. *NVIDIA Developer SDK 10*, February 2008.

[2] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses: the see-through interface. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 73–80, New York, NY, USA, 1993. ACM.

[3] S. Bruckner and E. Gröller. Style transfer functions for illustrative volume rendering. *Computer Graphics Forum*, 26(3):715–724, 2007.

[4] P. Brunet, R. Scopigno, M. C. Sousay, and J. W. Buchananz. Computer-generated graphite pencil rendering of 3d polygonal models. *Computer Graphics Forum*, 18(3):195–207, 1999.

[5] K. Bürger, F. Ferstl, H. Theisel, and R. Westermann. Interactive streak surface visualization on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, 15:1259–1266, 2009.

[6] B. Csebfalvi, L. Mroz, H. Hauser, A. König, and E. Gröller. Fast visualization of object contours by non-photorealistic volume rendering. In *Proceedings of Eurographics*, 2001.

[7] J. Diepstraten, D. Weiskopf, and T. Ertl. Transparency in interactive technical illustrations. *Computer Graphics Forum*, 21:2002, 2002.

[8] D. Ebert and P. Rheingans. Volume illustration: non-photorealistic rendering of volume models. In *VIS '00: Proceedings of the conference on Visualization '00*, pages 195–202, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[9] B. Freudenberg, M. Masuch, and T. Strothotte. Walk-through illustrations: Frame-coherent pen-and-ink style in a game engine. In *In Proceedings of Eurographics 2001*, pages 184–191, 2001.

[10] C. Garth, H. Krishnan, X. Tricoche, T. Tricoche, and K. I. Joy. Generation of accurate integral surfaces in time-dependent vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1404–1411, 2008.

[11] C. Garth, X. Tricoche, T. Salzbrunn, and G. Scheuermann. Surface techniques for vortex visualization. In *Proceedings Eurographics - IEEE TCVG Symposium on Visualization*, May 2004.

[12] A. Gooch, B. Gooch, P. Shirley, and E. Cohen. A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 447–452, New York, NY, USA, 1998. ACM.

[13] B. Gooch and A. A. Gooch. *Non-Photorealistic Rendering*. A. K. Peters Ltd., 2001.

[14] G. Gorla, V. Interrante, and G. Sapiro. Texture synthesis for 3d shape representation. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):512–524, 2003.

[15] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. H. Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005.

[16] A. Hertzmann. Introduction to 3d non- photorealistic rendering: Silhouettes and outlines. In *Non-Photorealistic Rendering (SIGGRAPH 99 Course Notes)*, 1999.

[17] J. P. M. Hultquist. Constructing stream surfaces in steady 3d vector fields. In A. E. Kaufman and G. M. Nielson, editors, *Proceedings of IEEE Visualization 1992*, pages 171 – 178, Boston, MA, 1992.

[18] T. Isenberg, B. Freudenberg, N. Halper, S. Schlechtweg, and T. Strothotte. A developer's guide to silhouette algorithms for polygonal models. *IEEE Comput. Graph. Appl.*, 23(4):28–37, 2003.

[19] T. Judd, F. Durand, and E. H. Adelson. Apparent ridges for line drawing. *ACM Trans. Graph.*, 26(3):19, 2007.

[20] G. Kindlmann, R. Whitaker, T. Tasdizen, and T. Möller. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *Proceedings of IEEE Visualization 2003*, pages 513–520, October 2003.

[21] H. Krishnan, C. Garth, and K. Joy. Time and streak surfaces for flow visualization in large time-varying data sets. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1267–1274, Oct. 2009.

[22] J. Krüger, J. Schneider, and R. Westermann. ClearView: An interactive context preserving hotspot visualization technique. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization 2006)*, 12(5), September-October 2006.

[23] R. S. Laramee, C. Garth, J. Schneider, and H. Hauser. Texture advection on stream surfaces: A novel hybrid visualization applied to CFD simulation results. In *Proc. Eurovis 2006 (Eurographics / IEEE VGTC Symposium on Visualization)*, pages 155–162, 2006.

[24] R. S. Laramee, J. J. van Wijk, B. Jobard, and H. Hauser. ISA and IBFVS: Image space-based visualization of flow on surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):637–648, 2004.

[25] H. Löffelmann, L. Mroz, and E. Gröller. Hierarchical streamarrows for the visualization of dynamical systems. In W. Lefer and M. Grave, editors, *Proceedings of the 8th Eurographics Workshop on Visualization in Scientific Computing*, pages 203–211, 1997.

[26] H. Löffelmann, L. Mroz, E. Gröller, and W. Purgathofer. Stream arrows: enhancing the use of stream surfaces for the visualization of dynamical systems. *The Visual Computer*, 13(8):359 – 369, 1997.

[27] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. Real-time hatching. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, page 581, New York, NY, USA, 2001. ACM.

[28] T. Saito and T. Takahashi. Comprehensible rendering of 3-d shapes. *SIGGRAPH Comput. Graph.*, 24(4):197–206, 1990.

[29] T. Schafhitzel, E. Tejada, D. Weiskopf, and T. Ertl. Point-based stream surfaces and path surfaces. In *Proc. Graphics Interface 2007*, pages 289–296, 2007.

[30] G. Scheuermann, T. Bobach, H. Hagen, K. Mahrous, B. Hamann, K. Joy, and W. Kollmann. A tetrahedra-based stream surface algorithm. In *Proc. IEEE Visualization '01 Conference*, pages 151–158, 2001.

[31] M. C. Sousa, K. Foster, B. Wyvill, and F. Samavati. Precise Ink Drawing of 3D Models. *EUROGRAPHICS2003*, 22(3):369–379, Sept. 2003.

[32] T. Strothotte and S. Schlechtweg. *Non-Photorealistic Computer Graphics*. Morgan Kaufmann, 2002.

[33] J. van Wijk. Implicit stream surfaces. In *Proceedings of IEEE Visualization '93 Conference*, pages 245–252, 1993.

[34] W. von Funck, T. Weinkauf, H. Theisel, and H.-P. Seidel. Smoke surfaces: An interactive flow visualization technique inspired by real-world flow experiments. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1396–1403, 2008.

[35] D. Weiskopf and T. Ertl. A hybrid physical/device-space approach for spatio-temporally coherent interactive texture advection on curved surfaces. In *GI '04: Proceedings of Graphics Interface*, pages 263–270. Canadian Human-Computer Communications Society, 2004.

[36] D. Weiskopf, T. Schafhitzel, and T. Ertl. Real-time advection and volumetric illumination for the visualization of 3d unsteady flow. In *Proc. Eurovis (EG/IEEE TCVG Symp. Vis.)*, pages 13–20, 2005.